# Meta-learning Sparse Implicit Neural Representations

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Implicit neural representations are a promising new avenue of representing general signals by learning a continuous function that, parameterized as a neural network, maps the domain of a signal to its codomain; the mapping from spatial coordinates of an image to its pixel values, for example. Being capable of conveying fine details in a high dimensional signal, unboundedly of its domain, implicit neural representations ensure many advantages over conventional discrete representations. However, the current approach is difficult to scale for a large number of signals or a data set, since learning a neural representation—which is parameter heavy by itself—for each signal individually requires a lot of memory and computations. To address this issue, we propose to leverage a meta-learning approach in combination with network compression under a sparsity constraint, such that it renders a well-initialized sparse parameterization that evolves quickly to represent a set of unseen signals in the subsequent training. We empirically demonstrate that meta-learned sparse neural representations achieve a much smaller loss than dense meta-learned models with the same number of parameters, when trained to fit each signal using the same number of optimization steps.

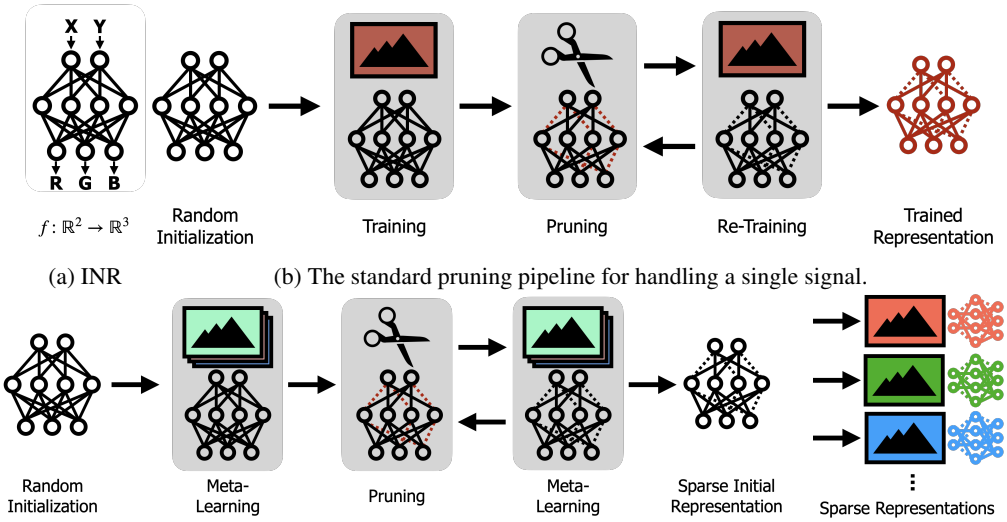
## 1 Introduction

An explosively growing line of research on implicit neural representations (INRs)—also known as *coordinate-based* neural representations—studies a new paradigm of signal representation: Instead of storing the signal values corresponding to the coordinate grid (e.g., pixels or voxels), we train a neural network with continuous activation functions (e.g., ReLU, sinusoid) to approximate the coordinate-to-value mapping [29, 25, 3]. As the activations are continuous, the trained INRs give a *continuous* representation of the signal. The continuity of INRs brings several practical benefits over the conventional discrete representations, such as providing an out-of-the-box method for superresolution or inpainting tasks [2, 35, 23, 34], or having its number of parameters not strictly scaling with the spatial dimension and/or spatial resolution of the signal [29, 25, 3, 8, 1, 32, 30]. Furthermore, as INRs take a form of trainable models, they are readily amenable to the idea of "learning a prior" from a set of signals, which is to be utilized for various purposes including image generation [2, 35, 34] or view synthesis [38, 34, 33, 26].

Despite its advantages, this network-as-a-representation approach is difficult to scale to handle a large set of signals, as having a parameter-heavy neural network trained for each signal requires a lot of *memory* and *computations*. Existing approaches to address this problem can be roughly divided into three categories. The first category of works utilizes a neural network structure shared across the signals, which takes a latent code vector[1] as an input and modifies the INR output accordingly (e.g., [34, 2, 29]). While these methods only need to store a latent vector for each signal (along with

[1] a low dimensional vector generated by forwarding each signal through some encoder,

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.




(a) INR

(b) The standard pruning pipeline for handling a single signal.

(c) The proposed Meta-SparseINR procedure for handling multiple signals.

Figure 1: Illustrations of (a) an implicit neural representation, (b) the standard pruning algorithm [9] that prunes and retrains the model for each signal considered, and (c) the proposed Meta-SparseINR procedure to find a sparse initial INR, which can be trained further to fit each signal.

shared model parameters), they have a limited ability to adapt to signals falling outside the learned latent space; this may lead to a degraded representation capability on new signals. The second line of works adopt meta-learning approach to train an *initial INR* from which each signal can be trained to fit within a small number of optimization steps [33, 38]. Unfortunately, these methods do not reduce the memory required to store the neural representations for each signal. The work belonging to the third category [4] reduces the memory required to store the INR model parameters by uniformly quantizing the weights of each INR. The method, however, does not reduce the number of optimization steps needed to train for each signal, or utilize any information from other signals to improve the quantization. In this sense, all three approaches have their own limitations in providing both memory- and computation-efficient method to train INRs for both seen and unseen signals.

To address these limitations, we establish a new framework based on the *neural network pruning*, i.e., removing parameters from the neural representation by fixing them to zero (e.g., [28]). Under this framework, the goal is to learn *sparse* INRs for a set of signals (either previously seen or unseen), in a computationally efficient manner. To achieve this goal, we make the following contributions:

- We formulate this problem as finding a *sparse initial INR*, which can be trained to fit each signal within a small number of optimization steps. By considering such formulation, we eliminate the need to prune the INRs separately for every single signal, each of which typically requires a computationally heavy prune-retrain cycle (Fig. 1b). We give a detailed formulation in Section 3.
- To solve this newly formulated problem, we propose a pruning algorithm called Meta-SparseINR, which is the first pruning algorithm designed for the INR setups (up to our knowledge). Meta-SparseINR alternately applies (1) a meta-learning step to learn the INR that can be efficiently trained to fit each signal, and (2) a pruning step to removes a fraction of surviving parameters from the INR using the meta-trained weight magnitudes as a saliency score (Fig. 1c). See Section 4 for a complete description of the algorithm.

In Section 5, we validate the performance of the proposed Meta-SparseINR scheme on three natural/synthetic image datasets (CelebA [20], Imagenette [14], SDF [38]) using a widely used INR architecture (SIREN [34])[2]. In particular, we show that, when trained to fit seen/unseen signals with a small number of optimization steps, Meta-SparseINR consistently achieves a better PSNR than the baseline methods using similar number of parameters, which include the meta-trained dense INRs with narrower width [38] and random pruning. Furthermore, in Section 5.3, we provide results on an exploratory experiment which sheds a light on the potential limitations of *pruning-at-initialization* schemes for fitting multiple signals under the INR context.

[2]We also report additional experimental results on another INR architecture (FFN [37]) in Appendix B.

## 2 Related works

Instead of giving a general overview on the implicit neural representations and neural network pruning literature, we focus on describing the works that are more directly related to the main ideas of this manuscript. For an overview of two disciplines, we refer the readers to excellent surveys of Tewari et al. [39] and Hoefler et al. [12] on implicit neural representations and the neural network pruning, respectively.

**Fitting implicit neural representations for multiple signals.** As briefly discussed in Section 1, there are at least three different approaches to efficiently train the INRs for a large set of signals. The first approach is to have a neural network component shared across the signals, which uses a latent code (i.e., a low-dimensional vector generated from each signal) to modify the INR output. Sitzmann et al. [34] trains a hypernetwork to predict INR parameters from the latent code, to utilize the knowledge learned from the signal set to fit each signal better. Other works [2, 25, 29] constructs an INR architecture that takes a concatenation of latent code and coordinate vector as an input, to predict the signal values corresponding to the each signal. This approach has advantages in terms of both memory and computation: in terms of parameters, the per-signal memory cost equal to the size of the latent vector, and the per-signal training cost is total training cost of the shared network, divided by the number of samples. These methods, however, have limitations in expressing the signals that has not been observed during the training phase, as pointed out in [38]. The second approach is to meta-learn the initial INR from which each signal can be efficiently fit using a small number of gradient descent steps. Sitzmann et al. [33] proposes an algorithm based on this approach to learn the signed distance function (SDF) corresponding to 2D/3D shapes, and shows that such pre-training by meta-learning significantly reduces the number of optimization steps required to encode new signals. Tancik et al. [38] proposes a similar algorithm that works well on a wider range of representation tasks, including 2D image regression. We note that, despite its computational benefits, this approach does not provide any benefit in terms of the parameter-efficiency. The third approach is compress each INR after training. Dupont et al. [4] propose one such method based on the weight quantization: They first fit INRs for each signal via extensive training. Then, they quantize the weights from 32-bit precision to 16-bit precision. This method does not provide any advantage in terms of computations, but can potentially be combined with other approaches. The approach taken in the present paper is most related to the second line of works [33, 38]; indeed, one of our main baselines will be the method of [38] used to train dense but narrower INRs.

**Training from sparse initial models.** Recent studies [6, 17, 22] have shown that one can find a sparse subnetwork (i.e., a pruned network) of a randomly initialized model that can closely achieve the performance level of the original dense model when trained from scratch. This finding has brought a surge of interest toward developing algorithms that can find such "winning ticket" subnetworks by masking the original models using various notions of saliency scores (e.g., [42, 36]). Although the research direction has seen a great progress overall, it has been less studied how one can find the best subnetwork when there are multiple datasets at hand. A closely related work is the paper by Morcos et al. [27], which shows that the winning ticket discovered from a dataset can be transferred to a highly relevant dataset. In this paper, the aim is to find a sparse initial model that performs well on *a set of different tasks* (which may be fitting a set of images/3D shapes in this context) when trained further on each task. We note, however, that we consider a slightly different setup than the previous works: We do not restrict ourselves to the models generated by pruning a randomly initialized model, and instead prune a meta-trained model. Indeed, our observation in Section 5.3 suggests that the models generated by pruning a randomly initialized model may not be able to significantly outperform dense and narrow baselines, in the INR context.

**Meta-learning & network pruning.** One of the earliest papers considering the combination of meta-learning with neural network pruning is the work by Liu et al. [21]. In [21], the goal is to train a *PruningNet* which generates a set of channel-pruned architecture and corresponding weights, satisfying the layerwise sparsity requirements given as an input. A more recent paper by Tian et al. [41] is slightly more related to our setting: In [41], the goal is to improve the generalization property of few-shot meta-learning algorithms, without any considerations on the memory or computation limitations; the proposed algorithm alternatively removes and restores the connections from the meta-learned model, as in DSD [10] or SDNN [15]. Unlike these works, the primary aim of this paper is to provide a method to *compress* a set of neural networks; we use meta-learning as a tool to generate a shared model that can be compressed and adapted to each signal. Despite its practical importance, up to our knowledge, we are the first to address this problem.

## 3 A framework for efficiently learning sparse neural representations

The goal of this paper is to develop a memory- and computation-efficient framework for learning and storing the implicit neural representations for a large number signals. We formulate this problem as finding a well-initialized sparse subnetwork structure; the sparse neural representation for each signal can be directly trained from this sparse subnetwork, without requiring a further pruning by computation-heavy prune-retrain cycle (see, e.g., [9]) for each individual signal.

**Formalisms.** For a formal description, we introduce some notations: Let $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$ be a set of $N$ target signals that we want to approximate. Each signal $T_j : \mathbb{R}^k \to \mathbb{R}^m$ is a function that maps a $k$-dimensional coordinate vector to an $m$-dimensional signal value. For example, a two-dimensional RGB image can be represented by the signal function $T : \mathbb{R}^2 \to \mathbb{R}^3$ with $T(x, y) = (R, G, B)$, and the occupancy of a 3D object (i.e., whether a point in 3D space in occupied by the object) can be represented by the coordinate-to-occupancy function $T : \mathbb{R}^3 \to \{0, 1\} \subset \mathbb{R}$. We assume that we have an access to data pairs $\{\mathbf{x}_i, T_j(\mathbf{x}_i)\}_{i\in\mathcal{I}}$ for each signal $j \in \{j \in 1, \ldots, N\}$, where $\{\mathbf{x}_i\}_{i\in\mathcal{I}}$ denotes the finite set of predefined input coordinate vectors indexed by $\mathcal{I}$. For two-dimensional $256 \times 256$ images, for instance, the input coordinate vectors may be 2D vectors belonging to the grid $\{0, {}^1/_{255}, {}^2/_{255}, \ldots, {}^{254}/_{255}, 1\}^2$.

We use the implicit neural representation $f(\cdot; \theta) : \mathbb{R}^k \to \mathbb{R}^m$ with a trainable parameter vector $\theta \in \mathbb{R}^d$ to approximate each signal. For SIRENs [34], $f(\cdot; \theta)$ will be a multi-layer perceptron (MLP) with sinusoidal activation functions; for Fourier feature networks (FFN [37]), $f(\cdot; \theta)$ will be an MLP with random Fourier feature encoders and ReLU activations. Now, the *representation risk* of approximating $j$-th signal with a parameter $\theta$ can be written as

$$\mathcal{L}_j(\theta) = \sum_{i\in\mathcal{I}} \|f(\mathbf{x}_i; \theta) - T_j(\mathbf{x}_i)\|_2^2. \tag{1}$$

Note that, while we are using the squared loss for a simple presentation, any other loss can be used (e.g., $\ell_1$ or even logistic loss for classification-like problems). Under typical scenarios without any sparsity considerations, one optimizes the parameters independently for each signal. In other words, we solve the optimization problem

$$\min_{\theta_1,\ldots,\theta_N\in\mathbb{R}^d} \frac{1}{N}\sum_{j=1}^{N} \mathcal{L}_j(\theta_j) = \frac{1}{N}\sum_{j=1}^{N} \left( \min_{\theta_j\in\mathbb{R}^d} \mathcal{L}_j(\theta_j) \right), \tag{2}$$

that can be decomposed as $N$ subproblems. In this case, the total number of parameters is $dN$.

**Sparse subnetwork and initialization.** To reduce the number of parameters, we consider finding a shared sparse subnetwork structure having at most $\kappa < d$ connections, so that the total number of parameters is reduced to $\kappa N$. More formally, the subnetwork structure can be expressed as a binary *mask* $M \in \{0, 1\}^d$ with $\|M\|_0 \leq \kappa$, which is applied to the neural representation by taking an element-wise product with the parameter, i.e., $f(\mathbf{x}; M \odot \theta)$. We hope to find a zero-one mask vector $M$ that solves the optimization

$$\min_{\substack{M\in\{0,1\}^d \\ \|M\|_0\leq\kappa}} \min_{\theta_1,\ldots,\theta_N\in\mathbb{R}^d} \frac{1}{N}\sum_{j=1}^{N} \mathcal{L}_j(M \odot \theta_j) = \min_{\substack{M\in\{0,1\}^d \\ \|M\|_0\leq\kappa}} \frac{1}{N}\sum_{j=1}^{N} \left( \min_{\theta_j\in\mathbb{R}^d} \mathcal{L}_j(M \odot \theta_j) \right). \tag{3}$$

The problem (3), however, needs to be refined for practical considerations. To solve each subproblem in Eq. (3) without too many gradient computations, we also require a good initialization. Indeed, it has been known that without a well-chosen initialization, the optimization of sparse networks often takes significantly longer [19], or even fail to achieve the performance level that can be achieved with a well-chosen one [6, 18].

To this end, we augment the problem (3) with the notion of *optimized initialization* as follows. Let $\theta_j^{(t)}(M; \theta^{(0)}) \in \mathbb{R}^d$ denote the parameter that has been attained by taking $t$ steps of gradient-based optimization (e.g., SGD or Adam) to fit the target signal $T_j$, on the subnetwork with an initial value $\theta^{(0)}$ and masked by $M$. Given some steps-to- budget $t$, we want to find a pair of mask $M \in \{0, 1\}^d$ and the initial value $\theta^{(0)} \in \mathbb{R}^d$ that jointly solves

$$\min_{\theta^{(0)}\in\mathbb{R}^d} \min_{\substack{M\in\{0,1\}^d \\ \|M\|_0\leq\kappa}} \frac{1}{N}\sum_{j=1}^{N} \mathcal{L}_j \left( M \odot \theta^{(t)}(T_j, \theta^{(0)}) \right). \tag{4}$$

**Algorithm 1** Meta-SparseINR: Meta-learning Sparse Implicit Neural Representations

**Require:** Signal set $\mathcal{T} = \{T_1, \dots, T_N\}$, desired sparsity $\kappa$, per-iteration pruning ratio $\gamma$,
number of pre-/re-training steps $\tau, \tilde{\tau}$, number of inner SGD steps $t$,
outer loop step size $\beta$.

Initialize $\theta \in \mathbb{R}^d$ using the standard initialization scheme.
Initialize mask $M = (1, 1, \dots, 1)$ of length $d$. ▷ Begin from the unpruned network.
$\theta \leftarrow \texttt{MAML}_{t,\beta}^{(\tau)}(\theta; \mathcal{T})$. ▷ Pretraining. See Section 4.1 for details.
**while** $\|M\|_0 > \kappa$ **do**
  $\mathbf{s}_i = M_i \cdot |\theta_i|, \quad \forall i \in \{1, \dots, d\}$. ▷ Compute the magnitude score.
  $\tilde{\mathbf{s}} = \text{SortDescending}(\mathbf{s})$
  $M_i \leftarrow \mathbf{1}\{\mathbf{s}_i \geq \tilde{\mathbf{s}}_{\gamma \cdot \|M\|_0}\}, \quad \forall i \in \{1, \dots, d\}$ ▷ Prune connections by the magnitude.
  $\theta \leftarrow \texttt{MAML}_{t,\beta}^{(\tilde{\tau})}(\theta; M, \mathcal{T})$. ▷ Retraining.
**end while**

We propose a meta-learning approach to solve the problem (4) in Section 4. Before we move on, we note that the problem cannot be directly solved—without an architectural change—by simply *pooling* the data pairs from $N$ signals and applying standard pruning techniques (e.g., [9]). This is because we are considering the task of signal representation, instead of a classification (or similar). Under the signal representation setup, the data pairs from different signals share the coordinate input space $\{\mathbf{x}_i\}_{i \in \mathcal{I}}$. If data pairs from different signals are pooled, this leads to conflicting pairs of samples that have different output values for the same input.

# 4 Learning sparse neural representations with meta-learning

As described in Eq. (4), our goal is to find a well-initialized sparse implicit neural representation from which the neural representations for each signal can be efficiently trained. To achieve this goal, we take a meta-learning approach: We meta-learn an implicit neural representation that can quickly adapt to a set of signals, and then prune the model based on its weight magnitudes (see Algorithm 1). We call this procedure Meta-SparseINR (Meta-learning Sparse Implicit Neural Representation). For completeness, we provide preliminaries on the meta-learning procedure we use in Section 4.1, and give a full description of the Meta-SparseINR algorithm in Section 4.2.

## 4.1 Preliminaries: Meta-learning with neural networks

Meta-learning [40] aims to give models that can be trained efficiently on a given task, by utilizing a form of knowledge gained from training on a family of relevant but different tasks. This knowledge can be transferred in various ways, e.g., utilizing neural network structures with memory-augmentations [31]. In this section, however, we focus on describing the *model-agnostic* approach that explicitly trains model parameters to convey the learned knowledge (namely, MAML [5]), which we use as a part of our method. This is because (i) it allows us to use popular implicit neural representation architectures (e.g., [34, 37]) without modifications, and (ii) we will use the model parameters as a saliency score for pruning, as will be described in the following subsection.

While MAML is not originally proposed for the signal representation, we describe MAML using the INR setup and notations (introduced in Section 3) for simplicity: Given a set of signals $\mathcal{T} = \{T_1, \dots, T_N\}$, MAML [5] aims to learn model parameters that can adapt to each signal within a small number of gradient steps. To do so, one performs the following iterative optimization: At each *outer loop*, the learner draws one (or more) signal from the signal set. For now, suppose that this is the $j$-th iteration of the outer loop and only one signal has been drawn, which we denote by $T_j$. Then, the learner computes a $t$-step SGD-updated version of the parameter $\theta^{(t)}(T_j, \theta_j)$ from the current parameter $\theta_j$. Then, one updates the parameter $\theta_j$ by taking a stochastic gradient descent (SGD) with some outer step size $\beta$:

$$\theta_{j+1} \leftarrow \theta_j - \beta \cdot \nabla_{\theta_j} \mathcal{L}_j \left( \theta^{(t)}(T_j, \theta_j) \right), \tag{5}$$

where $\mathcal{L}_j$ denotes the representation risk computed for the signal $T_j$, as in Section 3. This ends a single outer step. We denote running $\tau$ steps of outer loops on the current parameter $\theta$, using the signal set $\mathcal{T}$, outer step size $\beta$, number of inner SGD steps $t$ by $\texttt{MAML}_{t,\beta}^{(\tau)}(\theta; \mathcal{T})$. In our method, we will also perform MAML on the sparse networks (masked by some $M$). In this case, we will denote the same procedure by $\texttt{MAML}_{t,\beta}^{(\tau)}(\theta; M, \mathcal{T})$.

### 4.2 Meta-SparseINR: Meta-learning sparse implicit neural representations

We now describe our method, Meta-SparseINR, to learn the sparse implicit neural representation that can be efficiently trained to fit each signals. In a nutshell, Meta-SparseINR operates by alternately meta-training the INR using MAML, and pruning the INR using the magnitude-based pruning. More specifically, Meta-SparseINR consists of three steps:

- **Step 1 (Meta-learning).** Given a set of signals and an INR randomly initialized with standard initialization schemes (e.g., as in [34] for SIRENs), we meta-learn the INR over the signals by running MAML for $\tau$ steps.
- **Step 2 (Pruning).** From the learned INR, we remove $\kappa\%$ of surviving connections with smallest weight magnitudes. We use the *global* magnitude criterion, i.e., we apply a single magnitude threshold for every layers instead of having thresholds for each layer.
- **Step 3 (Retrain and repeat).** We retrain the pruned INR by running MAML for $\tilde{\tau}$ steps; $\tilde{\tau}$ required to converge to the desired performance level may often be smaller than $\tau$ used in Step 1. If the desired global sparsity level has not been met, go back to Step 2.

We refer the readers to Fig. 1c for a visual illustration of the Meta-SparseINR procedure, and to Algorithm 1 for a formal statement.

As we will validate through our experiments in Section 5, meta-learning is an essential component in Meta-SparseINR. More specifically, meta-learning steps play, at least, a twofold role: First, the weights given by the meta-learning can be trained to fit each signal with a much smaller number of optimization steps. This fact may not come as a big surprise, as the meta-learning objective (5) explicitly optimizes the *weight after optimizing for a few steps*. Indeed, without any pruning considerations, this point has been validated by Tancik et al. [38]. We observe that the same holds for sparse INRs, as we will observe in our main experiments in Section 5.2.

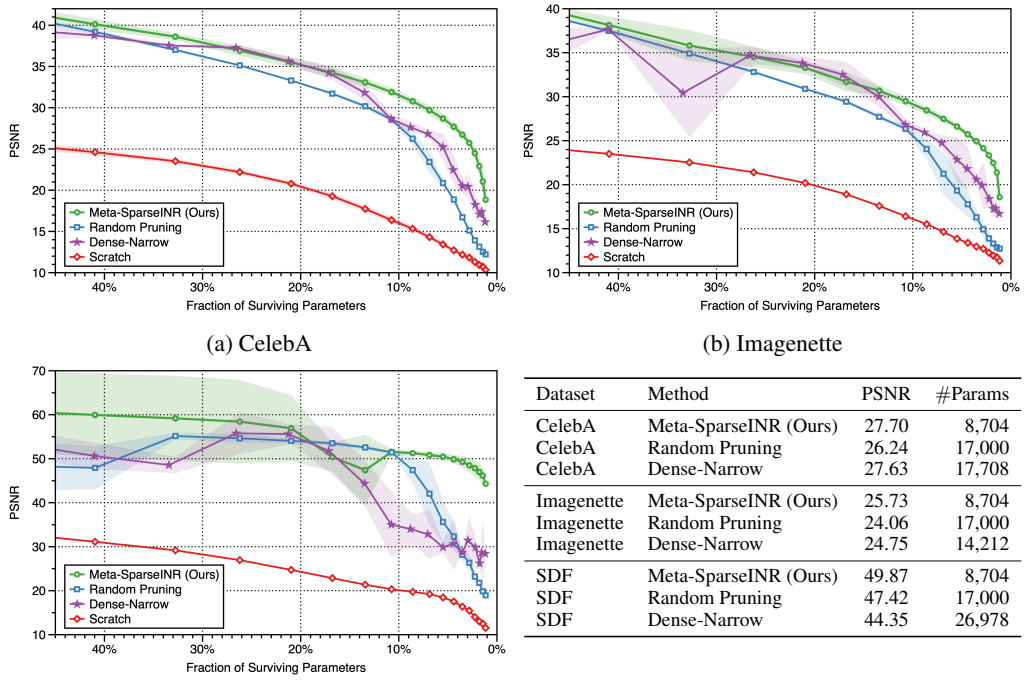
Second, the magnitude of the weights learned by meta-learning acts as a powerful *saliency score* for the pruning step (Step 2). Intuitively, the connections with a large meta-learned weight magnitudes are likely to have a relatively large weight magnitude after a $t$-step adaptation to a particular signal (with a small $t$). In this sense, the meta-learned weight magnitudes can be viewed as an *aggregation* of the weight magnitudes for each signal. As it is believed that weight magnitudes are an effective saliency criterion for the model trained for a single task (or signal, in this context), we expect the magnitudes of the meta-learned weights to be an effective saliency criterion for the signals, on average.[3] Indeed, we show in Section 5.2 that removing the connections with the smallest magnitude of the meta-learned weights outperforms both random pruning and dense, narrow INRs.

In addition to the second point, we note that we make an empirical observation which suggests that starting from the (meta-)trained weights may even be necessary for sparse models to outperform the dense, narrow models in the INR context. We give more details in Section 5.3.

## 5 Experiments

In this section, we experimentally validate the performance of the proposed Meta-SparseINR (see Algorithm 1) by measuring its performance on various 2D regression tasks. We describe the experimental setup and the baseline methods Section 5.1. We report the experimental results and discuss the observations in Section 5.2. In Section 5.3, we describe the experimental setup and results for an additional exploratory experiment; the experiment is designed to gauge the potential of "pruning-at-initialization" schemes to learn sparse INR that can be efficiently trained to fit each signal.

[3] Aside from this intuition, the magnitude criterion is much more computationally efficient than its competitors, such as Hessian-based ones [16, 11]; Hessian-based ones may require the evaluation of a third-order derivative, as we are using the meta-learning loss.




(a) CelebA

(b) Imagenette

(c) SDF

| Dataset | Method | PSNR | #Params |
|---|---|---|---|
| CelebA | Meta-SparseINR (Ours) | 27.70 | 8,704 |
| CelebA | Random Pruning | 26.24 | 17,000 |
| CelebA | Dense-Narrow | 27.63 | 17,708 |
| Imagenette | Meta-SparseINR (Ours) | 25.73 | 8,704 |
| Imagenette | Random Pruning | 24.06 | 17,000 |
| Imagenette | Dense-Narrow | 24.75 | 14,212 |
| SDF | Meta-SparseINR (Ours) | 49.87 | 8,704 |
| SDF | Random Pruning | 47.42 | 17,000 |
| SDF | Dense-Narrow | 44.35 | 26,978 |

(d) Comparison table

Figure 2: Comparison of the average PSNR of meta-learned SIRENs after 100 steps of per-signal training. (a–c) PSNR plots for CelebA, Imagenette, and SDF. (d) Comparing the Meta-SparseINR having 8,704 parameters with baselines achieving a similar level of PSNR. Shaded region: Mean $\pm$ one standard deviation (three seeds).

### 5.1 Experimental setup

We evaluate the performance of the proposed Meta-SparseINR under the following two-phase scenario: We are given a set of signals with training and validation splits. During the training phase, we meta-learn the initial INR from the training split of signals. During the test phase, we randomly draw 100 signals from the validation split of signals, and train for 100 steps using the full-batch Adam to fit the INR on each signal.[4] We report the average performance (PSNR) of the model over these 100 samples. We average over three seeds for all experiments in the following subsection.

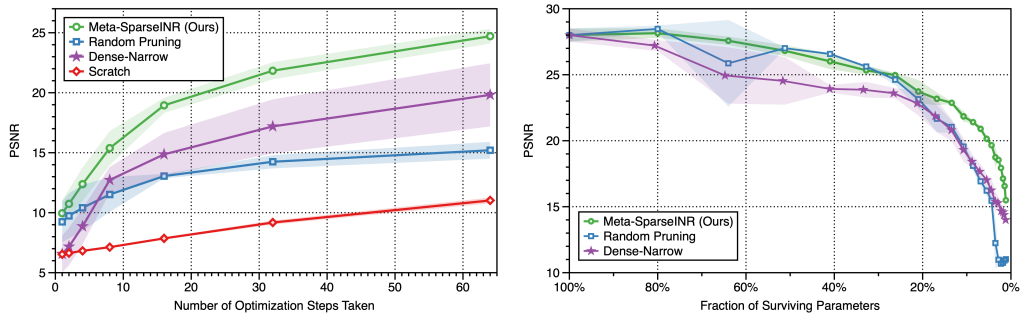
**Datasets.** We focus on 2D image regression tasks using three image datasets with distinct characteristics: CelebA (face data [20]), Imagenette (natural image [14]), and 2D SDF (geometric pattern [38]). We use the default training and validation splits. All images are resized and cropped to have the resolution of $178 \times 178$.

**Models.** As our base INR, we use multi-layer perceptrons (MLPs) with sinusoidal activation functions (SIREN [34]), having four hidden layers with 256 neurons in each layer. This configuration is popularly used for the 2D image regression task in previous works, e.g., [34, 38]. In Appendix B, we report additional experimental results on the another widely used class of architectures: MLPs using random Fourier features (FFN [37]).

**Method & baselines.** We evaluate the proposed Meta-SparseINR along with three baselines.

- *Meta-SparseINR (ours)*: The method proposed in this paper (see Algorithm 1). Throughout the experiments, we prune 20% of the weight parameters in every pruning phase (i.e., $\gamma = 0.2$).
- *Random Pruning*: Same as Meta-SparseINR, but we randomly prune the INR parameters with uniform probability, instead of using the magnitude criterion.
- *Dense-Narrow*: We meta-learn the dense neural representation that has a narrower width than the original INR, as in [38]. The widths are selected to ensure that each sparse INR has a dense-narrow counterpart with an equal or slightly larger number of total parameters.

[4] We ensure that we use same 100 signals for our method and the baselines.




(a) PSNR vs. number of optimization steps

(b) Two-step optimization

Figure 3: Fitting SIRENs meta-learned on CelebA to validation data with a smaller training budget. (a) PSNR trajectory for sparse models having 6,964 parameters and dense-narrow models having 7,152 parameters. (b) PSNR after two steps of training on each sample.

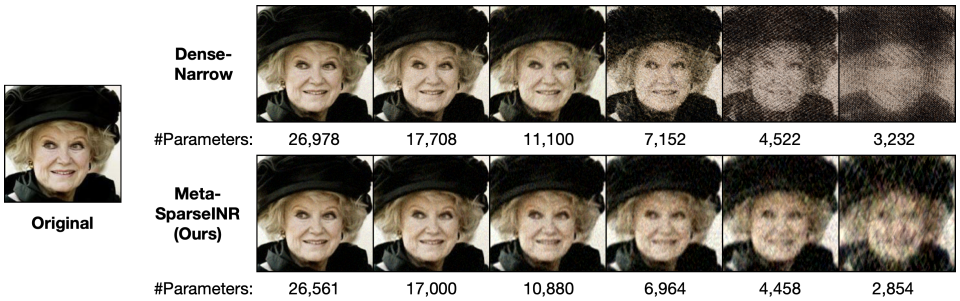



Figure 4: A visual comparison of meta-learned SIRENs after training for 100 steps to fit an image. **(Upper row)** Trained from a meta-learned dense and narrow INR, as in [38]. **(Lower row)** Trained from a sparse model learned by the Meta-SparseINR procedure. Sparse models retain more color information and give coarse patterns, while dense-narrow models are more monochromatic with less-structured patterns. We observed a similar behaviors in other examples (see Appendix C).

- *Scratch*: Same as Dense-Narrow, but we do not meta-learn the weights and use the weights initialized using the standard random initialization scheme (e.g., the one in [34] for SIRENs).

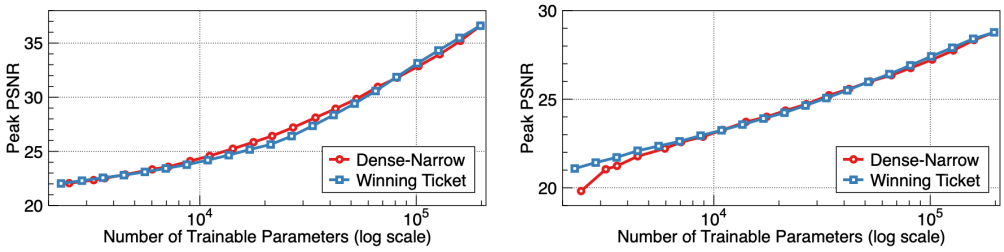
**Other details.** For other experimental details, including the data pre-processing steps and hyperparameters for the meta-learning phases, see Appendix A.

### 5.2 Main experiments

In Fig. 2, we observe that the proposed Meta-SparseINR achieves a consistent performance gain over the all baseline methods for all three datasets. We observe that the gain is larger for in the high-sparsity regime, i.e., for INRs having a much smaller number of parameters. In particular, for CelebA dataset on SIREN (Fig. 2a), Meta-SparseINR achieves the PSNR $\approx 26.73$ with only 6,964 parameters, whereas the strongest baseline (Narrow-Dense) achieves $\approx 20.52$ with slightly more 7,152 parameters; this is $\times 4.18$ difference in terms of the MSE loss. In Fig. 3, we observe that the same tendency holds when the training budget is smaller than 100 steps.

To better understand the effect of sparsity (as opposed to dense), we compare the distorted images generated by dense and narrow models with images generated by sparse models (Fig. 4). Here, we use the first image from the CelebA validation dataset. We observe that dense-narrow models tend to be more monochromatic and have finer patterns than wide-sparse models. Indeed, we observe a similar trend in other figures as well; see Appendix C for more images.

Lastly, we note that Meta-SparseINR outperforms the baselines in terms of the PSNR achieved for *training split* data as well. Indeed, the PSNR achieved for the training and validation splits differ only by a tiny margin. We report the results on the training split in Appendix D.




(a) SIREN (b) FFN

Figure 5: Size-PSNR tradeoff curves for fitting a $512 \times 512$ image using dense and narrow INRs (red circles) and wide-sparse INRs pruned at initialization using the "winning ticket" method (blue squares). In both plots, wide-sparse INRs do not perform noticeably well against dense and narrow ones, highlighting the importance of trained weights to enjoy the benefit of sparsity.

### 5.3 Pruning without trained initial weights

In this subsection, we ask whether it is possible to generate a sparse initial INR that can fit a set of signals efficiently, by simply pruning a randomly initialized INR directly.

**Setup.** We consider a simplified experiment on a *single* $512 \times 512$ natural image: We compare the post-training performances of (1) the sparse initial INRs generated by the iterative *winning ticket* method [6] using the global magnitude criterion, and (2) the randomly initialized dense INR with a narrower width. We select the winning ticket method, as it is known to outperform its competitors significantly in most cases [7]. We train for 10k steps with full batch, using Adam with learning rate $1.0 \times 10^{-4}$. For other experimental details (e.g., image, hyperparameters), see Appendix A.1.

**Result.** In Fig. 5, we report the peak PSNR achieved by both methods on SIREN (Fig. 5a) and FFN (Fig. 5b). We make two observations. First, we observe that the performance of both schemes drop consistently as we consider a smaller number of parameters. Second, none of two methods noticeably outperform the other. Together with the results in Section 5.2, the observations highlight the importance of the training step preceding the pruning step; meta-learning gives rise to a sparse structure that can be trained to a higher PSNR than the dense-narrow counterparts. This sheds a pessimistic light on the existence of an algorithm that can prune a randomly initialized INR to give a sparse INR that can be trained to fit mutliple signals better than a dense-narrow INR, as the best known pruning-at-initialization scheme does not provide any gain even for a single signal case.

## 6 Discussion & limitations

In this paper, we proposed a framework for an efficient training of implicit neural representations, in terms of memory and computation. We formulated the problem as learning a sparse INR that can be trained further to fit each signal within a small training budget, and proposed a first algorithm to solve the problem, called Meta-SparseINR; the proposed algorithm consistently outperforms the baselines in all experimental setups considered.

Adopting the neural network pruning approach, we have implicitly limited the discussion to methods that (1) generate sparse parameters, and (2) does not have any *shared parameters* available. While such properties can come advantageous for some applications (e.g., smaller inference cost), such restrictions are not necessary for improving the parameter efficiency. For instance, one may consider fitting the signals with INRs having a *sparse differences* from a dense, shared reference model; the memory cost to store the shared model will become negligibly small as the number of signals increases. Exploring the potential of such algorithms is a promising future direction.

A potential ethical side-effect of network compression algorithms is their disparate impact on various subgroups [13]. While such impact takes a form of disparate accuracy drop on subgroups, the impact can take a more nuanced and hard-to-detect form in the context of INR training. More specifically, we can think of two additional dimensions of disparate impact: the discrepancy in the training time required to arrive at a desired performance level, and *structurally-biased distortion* in the represented image, e.g., as in [24]. Such potential negative impacts on fairness should be carefully measured, disclosed, and mitigated before being considered for a public use.

# References


[1] M. Atzmon and Y. Lipman. SAL: Sign-agnostic learning of shapes from raw data. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[2] Y. Chen, S. Liu, and X. Wang. Learning continuous image representation with local implicit image function. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[3] Z. Chen and H. Zhang. Learning implicit fields for generative shape modeling. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[4] E. Dupont, A. Goliński, M. Alizadeh, Y. W. Teh, and A. Doucet. COIN: Compression with implicit neural representations. In *ICLR Neural Compression: From Information Theory to Applications Workshop*, 2021.

[5] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of International Conference on Machine Learning*, 2017.

[6] J. Frankle and M. Carbin. The lottery ticket hypothesis: finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

[7] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *International Conference on Learning Representations*, 2021.

[8] K. Genova, F. Cole, D. Vlasic, A. Sarna, W. T. Freeman, and T. Funkhouser. Learning shape templates with structured implicit functions. In *IEEE International Conference on Computer Vision*, 2019.

[9] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems*, 2015.

[10] S. Han, J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran, B. Catanzaro, and W. J. Dally. DSD: Dense-sparse-dense training of deep neural networks. In *International Conference on Learning Representations*, 2017.

[11] B. Hassibi and D. G. Stork. Second-order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems*, 1992.

[12] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. arXiv preprint 2012.00554, 2021.

[13] S. Hooker, A. Courville, G. Clark, Y. Dauphin, and A. Frome. What do compressed deep neural networks forget? arXiv preprint 1911.05248, 2019.

[14] J. Howard. Imagenette. URL https://github.com/fastai/imagenette/.

[15] X. Jin, X. Yuan, J. Feng, and S. Yan. Training skinny deep neural networks with iterative hard thresholding methods. arXiv preprint 1607.05423, 2016.

[16] Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, 1989.

[17] N. Lee, T. Ajanthan, and P. H. S. Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019.

[18] N. Lee, T. Ajanthan, P. H. S. Torr, and M. Jaggi. A signal propagation perspective for pruning neural networks at initialization. In *International Conference on Learning Representations*, 2020.

[19] N. Lee, T. Ajanthan, P. H. S. Torr, and M. Jaggi. Understanding the effects of data parallelism and sparsity on neural network training. In *International Conference on Learning Representations*, 2021.

[20] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *IEEE International Conference on Computer Vision*, 2015.

[21] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, T. K.-T. Cheng, and J. Sun. MetaPruning: Meta learning for automatic neural network channel pruning. In *IEEE International Conference on Computer Vision*, 2019.

[22] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019.

[23] I. Mehta, M. Gharbi, C. Barnes, E. Schechtman, R. Ramamoorthi, and M. Chandraker. Modulated periodic activations for generalizable local functional representations. arXiv preprint 2104.03960v1, 2021.

[24] S. Menon, A. Damian, S. Hu, N. Ravi, and C. Rudin. PULSE: Self-supervised photo upsampling via latent space exploration of generative models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[25] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[26] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamorthi, and R. Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, 2020.

[27] A. S. Morcos, H. Yu, M. Paganini, and Y. Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *Advances in Neural Information Processing Systems*, 2019.

[28] M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in Neural Information Processing Systems*, 1988.

[29] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[30] S. Peng, M. Niemeyer, L. Mescheder, and A. Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision*, 2020.

[31] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

[32] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. Scene representation networks: Continuous 3D-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, 2019.

[33] V. Sitzmann, E. R. Chan, R. Tucker, N. Snavely, and G. Wetzstein. MetaSDF: Meta-learning signed distance functions. In *Advances in Neural Information Processing Systems*, 2020.

[34] V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems*, 2020.

[35] I. Skorokhodov, S. Ignatyev, and M. Elhoseiny. Adversarial generation of continuous images. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[36] H. Tanaka, D. Kunin, D. L. K. Yamins, and S. Ganguli. Pruning neural networks without any data by iterative conserving synaptic flow. In *Advances in Neural Information Processing Systems*, 2020.

[37] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Advances in Neural Information Processing Systems*, 2020.

[38] M. Tancik, B. Mildenhall, T. Wang, D. Schmidt, P. P. Srinivasan, J. T. Barron, and R. Ng. Learned initializations for optimizing coordinate-based neural representations. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[39] A. Tewari, O. Fried, J. Thies, V. Sitzmann, S. Lombardi, K. Sunkavalli, R. Martin-Brualla, T. Simon, J. Saragih, M. Nie$\beta$ner, R. Pandey, S. Fanello, G. Wetzstein, J.-Y. Zhu, C. Theobalt, M. Agrawala, E. Shechtman, D. B. Goldman, and M. Zollhöfer. State of the art on neural rendering. *Eurographics*, 2020.

[40] S. Thrun and L. Pratt. *Learning to Learn*. Springer, 1998.

[41] H. Tian, B. Liu, X.-T. Yuan, and Q. Liu. Meta-learning with network pruning. In *European Conference on Computer Vision*, 2020.

[42] C. Wang, G. Zhang, and R. Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020.

# Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See the supplemental materials.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See the supplemental material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 5.2.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See the supplemental material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We use publicly available datasets, and have cited the reference.
   (b) Did you mention the license of the assets? [Yes] See the supplemental material.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See the supplemental material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We use only publicly available datasets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] See the provided code.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]